A PLUG-N-PLAY JAVASCRIPT WIDGET FOR ANNOTATING DOCUMENTS WITH MEDICAL JARGON

By

Tyler Anthony Pastor

A capstone project submitted for Graduation with University Honors

May 12, 2023

University Honors

University of California, Riverside

APPROVED

Dr. Paea LePendu
Department of Computer Science & Engineering

Dr. Richard Cardullo, Howard H Hays Jr. Chair
University Honors

ABSTRACT

A psychiatrist once asked our lab if it was possible to annotate their medical notes so that it's easier for his patients to understand the medical jargon found throughout the document. I am building a dynamic JavaScript widget that meets these needs. My plug-n-play widget will take in a medical document and dynamically tag all medical key terms. Once the tagging process is complete, the widget displays a pop-up information box for each key term. The pop-up information box holds related meta-data for users to further their understanding of the medical notes. Currently, I am working towards computing the most relevant definitions and relationships to display within the pop-up information box. To make this possible, I plan to connect several libraries together: a tagger to identify terms, a set of programs to organize and retrieve definitions, and additional tools to enable approximate matching and ranking to get the best possible results. This work is part of a larger project, Lexi – the Medical AI, which is a robust medical knowledge graph with multiple applications and endpoints, such as my knowledge widget.

ACKNOWLEDGEMENTS

I would like to thank Dr. Paea LePendu and Kevin Ferrer for providing great guidance and assistance throughout the duration of this capstone. I would also like to thank PiLabs for encouraging me to explore new areas of computer science and better my programming abilities.

NATURE AND PURPOSE

During one of my research lab sessions, a psychiatrist asked us if there were any existing resources that could help his patients better understand the medical jargon found in his medical notes. Because medical notes often contain medical terms that are foreign to the average person, many patients find it difficult to understand the content that they are reading. Without proper annotations, patients are often left confused due to their lack of understanding of specific terms, causing them to look up information on their own. If confused patients do not confirm their findings with their medical provider, it could potentially add to issues regarding misinformation or further misunderstandings. In order to combat this issue, I have developed a JavaScript widget capable of annotating any existing medical document on the internet. The medical widget supplies the patients with annotations that define key medical terms on the page, clarify any of the patient's concerns, and provide a number of relationships regarding the key medical term. Providing medically trusted definitions, relationships, and other relevant information for the medical jargon found throughout the medical notes allows patients to obtain a clearer understanding of any medical inquiries they have. In order to create this widget, I have connected APIs, JavaScript libraries, and databases, such as the Unified Medical Language System (UMLS), to extract the supporting data and serve it to the user as a plug-n-play Chrome extension.

To answer the psychiatrist’s question, there have been attempts to close the knowledge gap between the patient and medical information. One of the existing solutions comes from the University of Michigan Library, where they developed a plain language medical dictionary that “translates common medical terms into plain language, using definitions that were created by the U.S. government” (Plain Language Medical Dictionary 2020). The plain language medical

dictionary is a static web application that retrieves definitions based on user input. While this is a great solution for patients, one underlying issue is that users are required to use this application alongside the medical document they are reading, causing them to continuously leave the document they are reading to look up the meaning of a term. However, with the medical widget's dynamic capabilities, the patient can find the definition of a medical term on the same page, reducing the complexity of the research process. Another great feature I found during my research had come from Wikipedia. Wikipedia developed a widget that uses a term tagging system that identifies key terms within an article and provides pop-up information boxes for each of the terms. The pop-up information boxes contain brief descriptions and visual aids for each respective key term. Since this widget is very useful in providing relevant information for each tagged term found in Wikipedia articles, I have implemented this feature into my medical widget to help patients not only learn more about what they are reading, but also immediately clear up any questions or concerns. Combining all the information from the resources that I've gathered, I employ some of the methods found in the plain language medical dictionary and Wikipedia's term tagging system in the medical widget. However, unlike the plain language medical dictionary and Wikipedia's tagging system, the medical widget offers a dynamic solution that allows users to plug the widget into any webpage of their choice.

To make the application more accessible for users, I wrapped the medical widget into a Chrome extension. Once the medical widget is applied to the desired webpage, all medical terms are tagged and highlighted. Each tagged medical term is provided with a pop-up information box, similar to Wikipedia's term tagging system. Some additional features that I had added to the information box includes a definition of the respective tagged medical term, a few medical terms related to it, and the type of medical relation. The plugin provides a fast, yet clear solution for

patients via the pop-up information boxes. To obtain the concept, definition, and relationship information for all medical terms, I have utilized the UMLS developed by the National Library of Medicine. However, due to the UMLS limitations, some key terms are not contained in the database, resulting in missing definitions and relationships. In the case that a medical term is not found in the UMLS, I have attached a hyperlink to each term that directs the patient to the respective Wikipedia article. This allows users to learn more information about any tagged medical terms and clear up any remaining misunderstandings. The overall objective of the widget is to provide a new and concise method to properly educate patients on any form of medical documentation.

RELATED PROJECTS

The medical widget is related to a food index application that I helped develop which involves collecting nutritional information data to help identify food swamps. Food swamps are "understood as regions with very limited or difficult access to supermarkets and healthful food choices exemplify challenging food environments, which are generally more common in low-income urban areas" (Vilar-Compte et al. 2021). The food index application evaluates the nutritional health of a food item using the Ofcom United Kingdom nutrient profiling model, food prices, and the distance to a food place given the bounds of a user location (Lobstein and Davies 2009). Users can use these evaluations to make better eating habits and help prevent health issues such as heart disease, obesity, and malnutrition. As I have researched and developed health related applications in the past, I believe creating the medical widget is an application that would positively impact patients. Like the food index application, the medical widget can assist people with understanding medically related concepts pertaining to maintaining or improving their health.

Furthermore, both applications utilize Application Programming Interfaces (API), which allows for communication between two services or libraries. This process is essential to control the flow of data from one endpoint or library to another. The medical widget and food index application also use APIs to extract data from outside sources to be used in different parts of the application. For example, the food index application utilizes the Google Places API to retrieve a group of food places found within specified boundaries. For this current project, the medical widget uses the BioPortal Annotator API to fetch a list of medical terms found in a medical document. The food index application also uses another form of data collection, known as web scraping, to collect menu item data from various fast food and restaurant chains. This data is stored in a MySQL database to allow ease of access when evaluating the nutritional health of a given food place. Similarly, the medical widget stores all UMLS data in a MySQL database which is used to grab supporting data for a given medical term, such as definitions, relationships, and concept identifiers. After the food index application calculates the nutritional value for a given food item using the nutrient profiling algorithms, the results are stored in a separate database so they can be referenced in later parts of the application. Since the data is collected already, the nutritional value for items from a given fast food place no longer needs to be recomputed each time it is found in the user's vicinity, which increases runtime speeds. In addition, because nutritional health data remains universal, no matter the location of the fast food place, the nutritional facts for a given food item do not change. This idea is similar to how the medical widget does not continuously need to recompute the definition of a given medical term based on the user’s location. Both applications store data independent of the location or any external circumstance because the data (in this case, definitions of medical terms) remains static for each use case.

Another related project that helped shape the functionalities of the medical widget is Lexi, the Medical AI. Lexi is a medical knowledge graph that consists of various endpoints and applications, one of those endpoints being my medical widget. Lexi takes input from users' medical questions via voice commands and provides concise, accurate responses using curated data from the UMLS. Similarly to how Lexi uses UMLS as a database, my medical widget also utilizes the UMLS to "process texts to extract concepts, relationships, or knowledge" (Bodenreider O. 2004). One of the key reasons for using this database is because the UMLS consists of a number of medical ontologies – an ontology being an extensive knowledge base - housing numerous relationships between concepts, definitions, identifiers, and more. Some valuable ontologies found in the UMLS include RxNorm, SNOMED Clinical Terms, and MedlinePlus. Extracting the desired definition for a specific medical term is not trivial because multiple definitions, all coming from different ontologies, exist within the UMLS. Knowing this, I have specifically chosen to extract data based on a few ontologies found in the UMLS. In addition to the UMLS database, the medical widget utilizes the BioPortal Annotator API. The BioPortal Annotator API extracts all medical terms found within a medical document and sends the list of extracted terms back to my medical widget. Similar to the UMLS, the Annotator API offers an extensive list of medical ontologies to choose from when extracting/tagging medical terms. In order to help prevent tagging terms that do not directly contain medical jargon or tagging medical terms that might not be found in the UMLS, I chose to search for medical terms using a smaller set of ontologies offered by the BioPortal Annotator API.

DESIGN AND METHODS

My medical widget consists of several different libraries that play key roles in providing patients with information they need to understand the medical jargon found in medical documents. These libraries include the UMLS, RESTful web services, BioPortal Annotator API, Wikipedia, and Chrome Extension Developer tools. API requests permit the libraries to transfer data to one another. This is a key aspect of my application because data is continuously being retrieved from one source and used in another. Before the medical terms can be highlighted using JavaScript tools, the BioPortal Annotator API must respond to the medical widget with a list of tagged medical terms that were found in a block of text. Similarly, the definitions, relationships, and concept information cannot be retrieved from the UMLS without first extracting the medical terms from the medical document. Due to these constraints, the medical widget must execute each library in a specific order to allow for the proper flow of data and produce reliable information to the user.

The first action that must be performed within the medical widget is to extract all paragraph elements found in the HTML document. This step requires JavaScript tools that enable data retrieval from a pre-existing webpage. Validation checks are placed after each paragraph element is queried. The medical widget must check that every paragraph element does not contain any child elements. This ensures that all paragraph elements only contain inner HTML text which is later used at the point of term extraction and during the process of text reassembly. It is important to note that the rest of the application's data transfer processes occur for a single paragraph element before continuing to the next paragraph element. Once each paragraph element has been retrieved and validated, the term tagging process ensues for the first paragraph element.

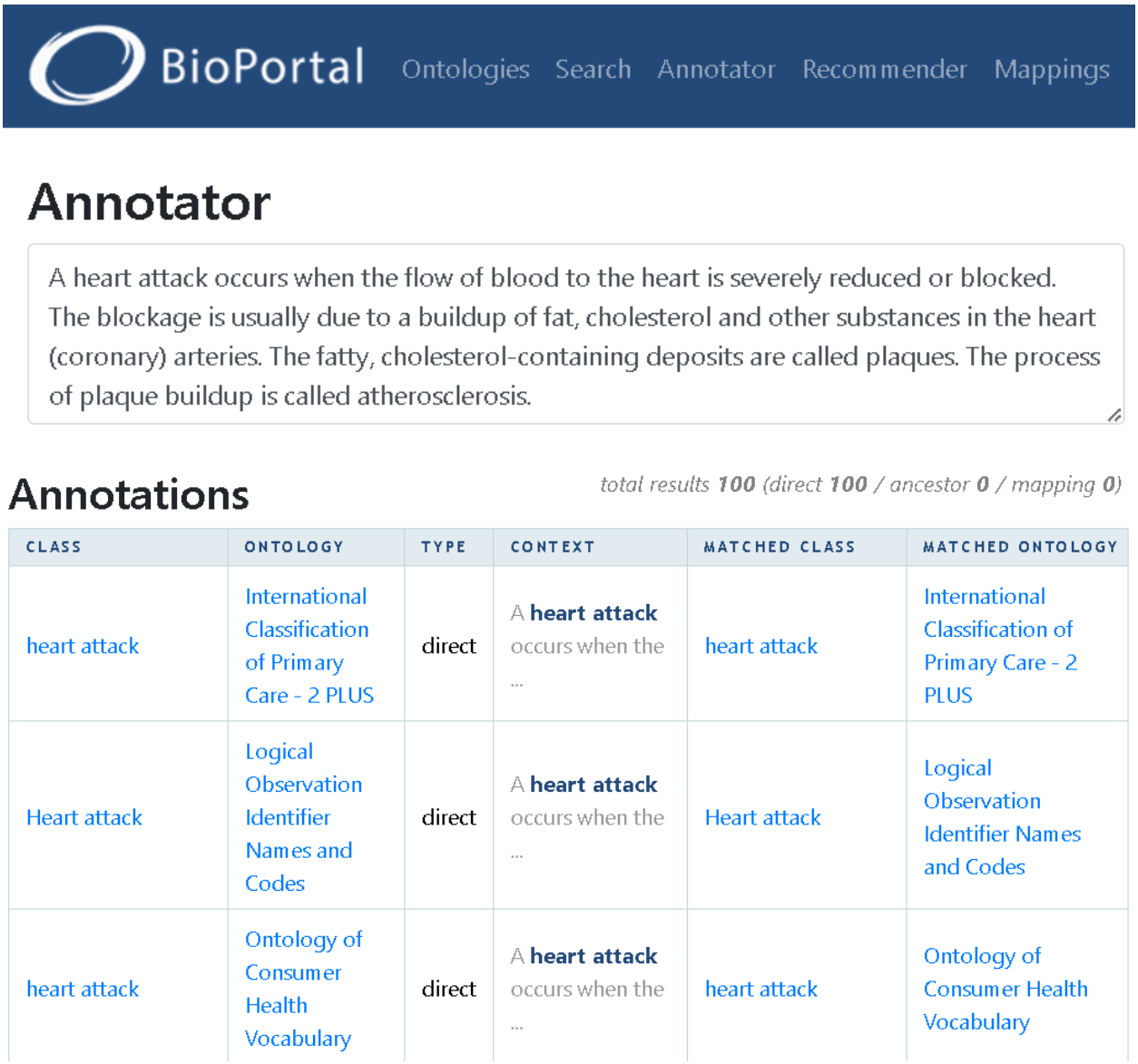

Figure 1. BioPortal Annotator Web API

As shown in the figure above, the term tagging process begins with the BioPortal Annotator API. The paragraph element's inner HTML text that was initially retrieved in the previous step is now used within the BioPortal Annotator API to extract all medically related terms. The figure above displays the public web version of the Annotator API which allows developers to insert a block of text, set given API parameters, and receive a table of all medical terms found within the text. In order to retrieve a list of tagged medical terms in the medical

widget, I utilized BioPortal's public API GET request route to the Annotator. The GET request takes in a base URL concatenated with the text to be annotated. The URL also contains the API parameters that can be manually set to produce different results. For example, I have set a number of parameters that cause the Annotator API to search for medical terms based on a smaller set of ontologies offered by BioPortal, match terms based on the longest instance (the term heart attack matches "heart attack" rather than "heart"), exclude number, and exclude synonyms. I chose to search for medical terms using a small set of ontologies because some ontologies found in BioPortal are not included in the UMLS. Due to this problem, a significant number of medical terms that have been tagged might not have supporting data in their pop-up information box. To help further remediate this issue, I specifically chose to exclude synonyms from being returned to the medical widget. Synonyms could not only lead to lacking supporting data, but also cause issues within the text reconfiguration phase of the medical widget. A list of tagged medical terms is returned to the medical widget in JavaScript Object Notation format. This allows the medical widget to easily extract each term and manipulate the list for future phases of the entire process. The figure above shows a snippet of the total results received from the Annotator API. Multiple instances of the term "heart attack" can be found in the list. The medical widget must remove duplicates for each term in the list to increase processing speeds and prevent tagging issues. Without removing duplicates, the same term would be used to locate and fetch related definitions, relationships, and concept information multiple times. This would severely slow the time it takes to inject my medical widget into a webpage a user is reading. Furthermore, terms would be tagged multiple times and interfere with the programming logic which would affect the text's sentence structure.

After removing all duplicate terms from the list, the medical widget begins separating the relative block of text into sections, where each tagged medical term provides a point of separation. Essentially, every piece of the text that does not contain a tagged medical term will be separated from the text and stored into a list to be used later. This process helps with reconfiguring the text once the tagged medical terms have been assigned specific CSS properties. Before adding the CSS properties, the medical widget creates an a-tag element which will contain the tagged medical term. This defines the tagged medical term as a hyperlink which redirects users to read more information about the term. As previously mentioned, some tagged medical terms might not be found within the UMLS which leaves the pop-up information empty and does not provide users with any supporting information. To deal with this issue, I have attached Wikipedia articles to each tagged medical term. Upon clicking on a tagged medical term, users will be redirected to the respective Wikipedia article to view more supporting information. Each tagged medical term is highlighted to distinguish from the rest of the text within the webpage. When a user hovers over a tagged medical term, it will be underlined to indicate which term the pop-up information box belongs to, in the case that two tagged medical terms fall next to each within the text. At this point, the remaining steps include extracting all related definitions, relationships, and concept data.

```
{
  "Definitions_Returned": 14,
  "Definitions": [
    {
      "CUI": "C0011849",
      "AUI": "A7570023",
      "SAB": "NCI",
      "STR": "Diabetes Mellitus",
      "DEF": "A metabolic disorder characterized by abnormally high blood sugar levels due to diminished production of insulin or insulin resistance/desensitization."
    },
    {
      "CUI": "C0011849",
      "AUI": "A7570023",
      "SAB": "NCI_NCI-GLOSS",
      "STR": "Diabetes Mellitus",
      "DEF": "A disease in which the body does not control the amount of glucose (a type of sugar) in the blood and the kidneys make a large amount of urine. This disease occurs when the body does not make enough insulin or does not use it the way it should."
    },
    {
      "CUI": "C0011849",
      "AUI": "A24672870",
      "SAB": "HPO",
      "STR": "Diabetes mellitus",
      "DEF": "A group of abnormalities characterized by hyperglycemia and glucose intolerance. [HPO:probinson]"
    },
```


Figure 2. Definition route's returned JSON data

I created a RESTful web service (API) that allows the medical widget to quickly access any necessary supporting data for a given medical term. The medical widget utilizes GET requests to receive different types of information from the UMLS. Two routes are used, one to grab related definition data and the other to grab relationship data respective to the tagged medical term. The figure above displays how the JSON data looks when the medical widget requests the definition data for the term "diabetes mellitus". The definition route returns the concept unique identifier (CUI), atomic unique identifier (AUI), ontology source (SAB), the given medical term (STR), and the term's definition (DEF). The definition route is the first GET request to be called within the medical widget because the relationship route requires the AUI to

retrieve a list of relationships for a given medical term. Error checking is implemented within this phase to prevent unnecessary look-ups and increase processing speeds. After the definition data is returned to the medical widget, it checks to see the number of definitions that have been returned. If the tagged medical term is not found in the UMLS, resulting in no definitions found, then the medical widget skips over the GET request used to grab the relationship data.

In order to grab the data from the UMLS, the RESTful web service uses Node JS, a JavaScript framework, to create a connection to the MySQL database that houses all the UMLS data. The MySQL database contains a number of tables that are referenced, manipulated, and joined by the RESTful web service to properly extract all supporting data in the correct format. The three tables referenced by the RESTful web service include the concept table (MRCONSO), the definition table (MRDEF), and the relationship table (MRREL). The figure below supplies a SQL script I created to combine some of the contents from both MRCONSO and MRDEF and the related output formatted into a table.

```
SELECT DISTINCT
d.CUI, d.AUI, d.SAB, c.STR, d.DEF
FROM MRDEF d, MRCONSO c
WHERE d.CUI IN
    (SELECT DISTINCT CUI
    FROM MRCONSO
    WHERE STR = 'heart'
    AND ISPREF = 'Y')
AND d.AUI = c.AUI
AND d.CUI = c.CUI;
```

	CUI	AUI	SAB	STR	DEF
1	C0153500	A7605131	NCI_NCI-GLOSS	Malignant Cardiac…	A rare can…
2	C0153500	A7605131	NCI	Malignant Cardiac…	A primary …
3	C0153957	A7625888	NCI	Benign Cardiac Ne…	A non-meta…
4	C0018787	A0066374	UWDA	Heart	Organ with…
5	C0018787	A7570969	NCI	Heart	A hollow o…
6	C0018787	A7570969	NCI_CDISC	Heart	A hollow m…
7	C0018787	A15475732	FMA	Heart	Organ with…
8	C0018787	A0066368	MSH	Heart	The hollow…
9	C0018787	A0480532	CSP	heart	hollow, mu…

Figure 3. SQL script used to fetch supporting data and respective table output

Combining the MRCONSO and MRDEF tables is necessary to retrieve all supporting information used by the medical widget because the data can not be found in one single table. Both MRCONSO and MRDEF contain some similar data sets, such as the CUI, AUI, and SAB, however, they differ when it comes to locating the term and definition in the same location. This

makes it difficult to search for definitions in MRDEF based on a given medical term because MRDEF does not contain any terms, only the related concept identifiers. Since MRCONSO contains both medical terms and related concept identifiers, the medical widget uses this data to fetch the definition data found in MRDEF, essentially combining the tables. The concept identifier data is used within the look-up SQL script to grab the definition for a given medical term. Similarly, the relationship route uses the atomic identifier data that is received from the definition route request to properly find all medical terms linked to the tagged medical term. As shown in the table above, the definition route returns the AUI for a given medical term which is utilized in the relationship route. The relationship route returns a new set of data used to provide supporting relationship data within the pop-up information box.

Once the supporting definitions and relationships are retrieved and stored in a list for each tagged medical term, the medical widget reconfigures the original structure of the block of text using the pieces of text that was stored in a previous phase and the newly created a-tag elements. Note that a placeholder element is used to continuously add pieces of text and a-tag elements as the list of tagged medical terms decreases. After a tagged medical term is used to fetch the supporting data and added to the placeholder element, it is then removed from the list of tagged medical terms and the process repeats with the following tagged medical term. Once the list of tagged medical terms is empty, the medical widget adds all supporting data to each respective tagged medical term. Several div-elements, pop-up information boxes, are created to store the definitions and relationships. Each div-element uses the same CSS properties in order to properly display or hide the pop-up information box when the user is hovering over a tagged medical term or not. At this point, the medical widget has completed all processes and created a better platform to inform users about medical jargon found in medical documents.

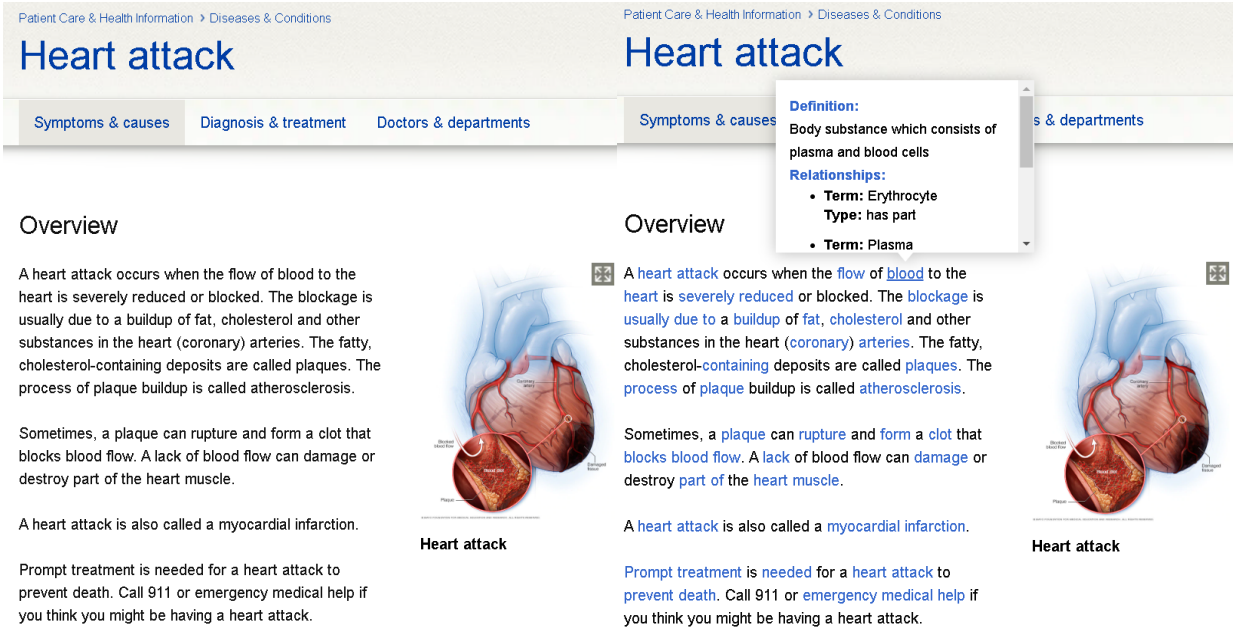

Figure 4. Before the medical widget is applied (on the left) and after it is applied (on the right)

The figure above showcases my medical widget in action on the Mayo Clinic website for information regarding heart attacks. Without the medical widget, a patient would have to scour the internet for answers regarding any misunderstandings they may have for a given medical term. This could lead to the spread of misinformation and potentially cause harm to those who are not careful as they read untrustworthy material on different webpages. My medical widget provides safe, accurate, and trustworthy data for users. Moreover, the medical widget conveniently offers this information directly on the web page the user is reading. As a result, the user no longer has to hassle with using outside sources as a frame of reference and continuously validate the information they find.

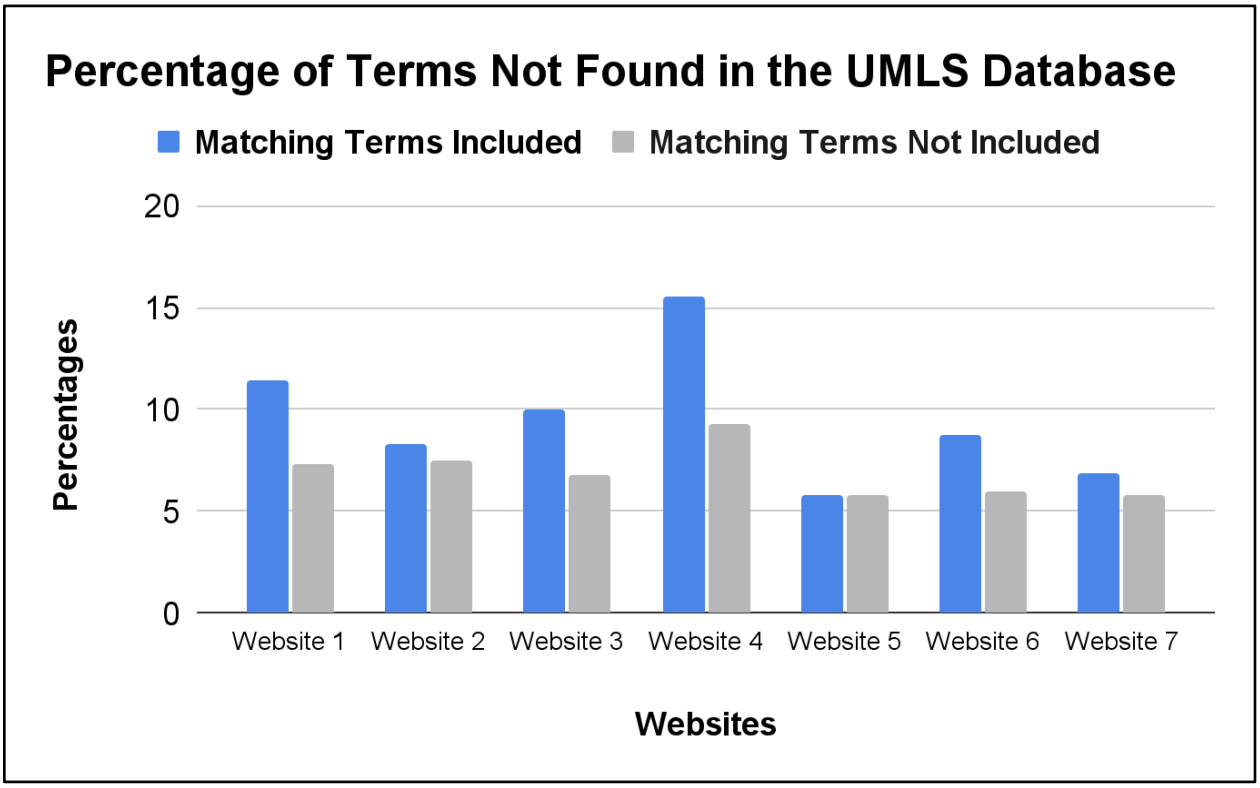

Figure 5. Graphic showcasing terms not found in the UMLS for each website

After completing all the functionalities of the medical widget, I began testing it on several different medical websites. Through numerous amounts of testing, I learned that on average, less than 7% of all medical terms tagged by the BioPortal Annotator API were not found in the UMLS. These tests were completed across 7 different websites, which is displayed in the chart above. Two lists were retrieved after completing a single test on a website. The first list is represented by the blue bar which includes all tagged medical terms that were not found in the UMLS, including duplicate terms. The second list is represented by the gray bar and is quite similar to the first list, however, all duplicate terms are removed from the list in order to provide an accurate representation of the percentage of terms not found in the UMLS. The top five most common terms not found in the UMLS that do not contain medical jargon are “do not”, “is a”,

“too much”, “ensure”, and “privacy policy”. These terms do not contain any medical jargon which shows that the BioPortal Annotator API is not properly filtering the tagged medical terms.

In the future, I would like to implement a system within the medical widget to confirm that a tagged medical term does in fact contain medical jargon. The UMLS fails to identify some medical terms which may distract from the patient’s ability to better their medical understanding. The terms “tylenol”, “squamous”, and “glumetza” were tagged by the Annotator API but were not found in the UMLS. In order to solve the medical widget’s limited data resource issues, I attached Wikipedia articles to each tagged medical term, as previously mentioned. I would like to expand my resources beyond the UMLS to prevent the user from needing to leave the medical document they are reading, and to be able to access all the information needed on just one page. Expanding these resources could allow the medical widget to use high-level supporting data that is easy to understand by the general public. Many of the definitions and relationships currently found in the UMLS contain medical jargon that may be difficult for patients to understand, which does not help patients clear up their misunderstandings. Also, I find it important to supply visual aids with the supporting text for patients. Images and diagrams can help explain certain concepts that might be difficult to understand through text.

My medical widget was developed to better educate patients by helping them understand their doctor’s medical notes. To make the process more accessible and convenient for patients, the medical widget is dynamically capable of being plugged into any web page of the user's choice. I developed the medical widget using several different libraries, all connected to one another to allow the proper transfer of data from backend to frontend. The backend of the application consists of MySQL, Node JS, the Unified Medical Language System (UMLS), RESTful Web Services, and the BioPortal Annotator API. These libraries are crucial to the

gathering and manipulating of medical data. The frontend of the application contains more JavaScript tools, CSS properties, and Wikipedia articles. The frontend of the application creates a seamless viewing experience and allows patients to easily view supporting information for a given medical term. Both the backend and frontend of the application is wrapped inside a Chrome extension which allows users to easily plug my medical widget into the webpage of their choice. All in all, this was a great experience as I learned more about medical relationships, data transfer, and data collection. I believe my application is applicable in useful real-world settings, like in the medical industry, for both patients and doctors.

REFERENCES

"Anxiety Disorders." *National Institute of Mental Health*, www.nimh.nih.gov/health/topics/anxiety-disorders. Accessed 28 Apr. 2023.

"Basic Information About Skin Cancer." *Center for Disease Control and Prevention*, www.cdc.gov/cancer/skin/basic_info/index.htm. Accessed 28 Apr. 2023.

Bodenreider O. The Unified Medical Language System (UMLS): integrating biomedical terminology. Nucleic Acids Res. 2004 Jan 1;32(Database issue):D267-70. doi: 10.1093/nar/gkh061. PubMed PMID: 14681409; PubMed Central PMCID: PMC308795.

"Depression." *American Psychological Association*, www.apa.org/topics/depression. Accessed 28 Apr. 2023.

"Diabetes." *Mayo Clinic*, www.mayoclinic.org/diseases-conditions/diabetes/symptoms-causes/syc-20371444. Accessed 28 Apr. 2023.

"Heart Attack." Mayo Clinic, 21 May 2022, www.mayoclinic.org/diseases-conditions/heart-attack/symptoms-causes/syc-20373106. Accessed 26 Apr. 2023.

Lobstein, T., and S. Davies. “Defining and Labelling ‘Healthy’ and ‘Unhealthy’ Food.” *Public Health Nutrition*, vol. 12, no. 3, Mar. 2009, pp. 331–40, https://doi.org/10.1017/S1368980008002541.

"Plain Language Medical Dictionary." *University of Michigan Library*, apps.lib.umich.edu/medical-dictionary/. Accessed 28 Apr. 2023.

"Stress." *World Health Organization*, www.who.int/news-room/questions-and-answers/item/stress. Accessed 28 Apr. 2023.

"Tylenol." *Drugs.Com*, www.drugs.com/tylenol.html. Accessed 28 Apr. 2023.

Vilar-Compte, Mireya, et al. “Urban Poverty and Nutrition Challenges Associated with Accessibility to a Healthy Diet: A Global Systematic Literature Review.” *International Journal for Equity in Health*, vol. 20, no. 1, Jan. 2021, p. 40, https://doi.org/10.1186/s12939-020-01330-0.

Whetzel PL, Noy NF, Shah NH, Alexander PR, Nyulas C, Tudorache T, Musen MA. BioPortal: enhanced functionality via new Web services from the National Center for Biomedical Ontology to access and use ontologies in software applications. Nucleic Acids Res. 2011 Jul;39(Web Server issue):W541-5. Epub 2011 Jun 14.

"Wikipedia." Wikipedia, Wikimedia Foundation, www.wikipedia.org/. Accessed 12 May 2023.